MP 5.5 An Out-of-Order Three-Way Superscalar Multimedia Floating-Point Unit

Alisa Scherer, Michael Golden, Norbert Juffa, Stephan Meier, Stuart Oberman, Hamid Partovi, Fred Weber

Advanced Micro Devices, Sunnyvale, CA

The AMD-K7™ floating point unit is implemented as an out-of-order coprocessor responsible for executing all x86 FPU, MMX™, and AMD 3DNow!™ instructions [1]. The FPU interfaces to the AMD-K7 core, which sends it instructions, load data, and guides the retirement of instructions. The FPU sends store data and completion status back to the core. Figure 5.5.1 shows a block diagram of the FPU. The FPU contains 2.4M transistors on a 10.5x2.6mm^2 die in a 0.25µm process. A micrograph of the FPU is shown in Figure 5.5.2.

FPU control consists of an in-order front-end that decodes and maps x86 instructions to internal execution ops. A central scheduler dispatches execution ops into execution pipes when their source operands are available. Pipe tracking logic reports completion status to the core. A retire queue holds all in-flight ops, maintains the register freelist, and updates architectural state as ops are retired by the core.

The front end decodes up to three x86 instructions per cycle. This involves first mapping them into three operand format internal execution ops with the stack relative-register references converted to absolute registers. Complex x86 FPU instructions (e.g. transcendentals) are received from the core directly as a series of microcoded ops which are easily mapped into FPU execution ops. Absolute register numbers of execution ops are renamed into physical register numbers, using a mapper for the source which provides the most recent physical register mapped to a given absolute register, and obtaining destination physical register numbers from the freelist. The last stage of the in-order front-end inserts renamed execution ops into a 36-entry scheduler.

Ops are issued from the scheduler when their source registers are ready and the required execution resources are available. Sources may come from the register file, be bypassed directly from one of three result buses, or in the case of memory operands, come from one of two load operand buses. Each source in the scheduler only has to snoop a maximum of three buses since it is determined ahead of time whether to snoop the three result buses or the two memory operand buses. Once an op is issued, it proceeds to read the register file and then enters the appropriate execution pipe. The scheduler employs a compaction scheme that allows space to be freed in the scheduler as ops are issued to the functional units.

On completion of an operation, the result is written to the destination register, and completion status is sent to the core, which enables the core to retire the op. The retire queue holds up to 72 speculative ops and is responsible for updating architectural state and placing the old destination registers back onto the freelist when the ops are retired.

The FPU contains three execution pipelines: add, multiply, and store. The add pipeline computes all x87 FP addition, subtraction, and compare operations, as well as MMX integer ALU operations and 3DNow! FP additions. The store pipeline processes true store operations, along with several special operations to support microcode routines.

The multiply pipeline computes all x87 FP multiplication, remainder, division, and square-root operations, MMX integer ALU and multiplication operations, and 3DNow! FP multiplications. The 76x76b multiplier employs radix-8 Booth encoding to generate 26 partial products in the first execution cycle (Figure 5.5.3). A binary tree of 4-to-2 compressors reduces partial products to two, after which a rounding constant is added through two parallel (3,2) carry-save adders. These results are carry-assimilated in the third cycle, and the result is chosen in the fourth cycle. Division and square root use a quadratically-converging multiplication-based algorithm, and they share the FP multiplier, forming the constraints on the dimensions of the multiplier. FP multiplication operations are able to fill unused cycles during a division or square root to provide maximum execution bandwidth. The latency and throughput of the various operations are shown in Table 5.5.1.

The floating-point register file holds 88 words, each 90b. The register file has five read ports and five write ports that operate simultaneously in a clock cycle. A pulsed write enable signal and flopped write data are driven directly to the register cells without any intervening logic. The register bit cell, Figure 5.5.4, uses pull-downs on both sides of the cross-coupled-inverter storage node to speed up the write without use of pMOS devices. This fast write permits write-through of write data to a read in the same cycle from the same register without special bypass circuitry. This contrasts with designs using single-ended writes which do not allow write-through, especially at lower voltages [2]. The register file is self-timed and self-resetting, and relies only on the falling edge of the clock to start the timing chain.

Each read port of the register file is separately enabled to reduce power dissipation when data is unchanged. Similarly, each functional unit is enabled only when performing true computation. Power is managed in the control queues by using valid bits to control conditional flip-flops, reducing power dissipation during periods without valid instructions.

The FPU uses a variation of the pulsed flip-flop as principal latching element [3]. In addition to its small Tsu + Tcq, this topology incorporates complex logic in its first stage using a dynamic pull-down network, a feature utilized throughout the FPU to improve timing of critical paths. Figures 5.5.5 and 5.5.6 show the enabled and the 4-way mux enabled flip-flops. The muxed flip-flop is functionally equivalent to the basic flop preceded by a multiplexer but improves critical path latency by up to 12%. Due to the dynamic nature of the first stage of the flip-flop, coupling to its inputs must be tightly controlled. To this end, a CAD tool determines the minimum allowable input signal strength based on the driver distance from the flip-flop.

Acknowledgements:
The authors acknowledge technical contributions of M. Roberts, J. Trull, M. Achenbach, J. Fan, M. Gulati, C. Keltcher, P. Lam, M. Siu, and J. Tseng.

References:
[1] Oberman, S., et al., "AMD 3DNow! Technology and the K6-2 Microprocessor," Proceedings of Hot Chips 10, pp. 245-254, Aug., 1998.
[2] Gieseke, B., et al., "A 600 MHz Superscalar RISC Microproces-sor with Out-of-Order Execution," ISSCC Digest of Technical Papers, pp. 176-177, Feb., 1997.
[3] Partovi, H., et al., "Flow-Through Latch and Edge-Triggered Flip-Flop Hybrid Elements," ISSCC Digest of Technical Papers, pp. 138-139, Feb., 1996.

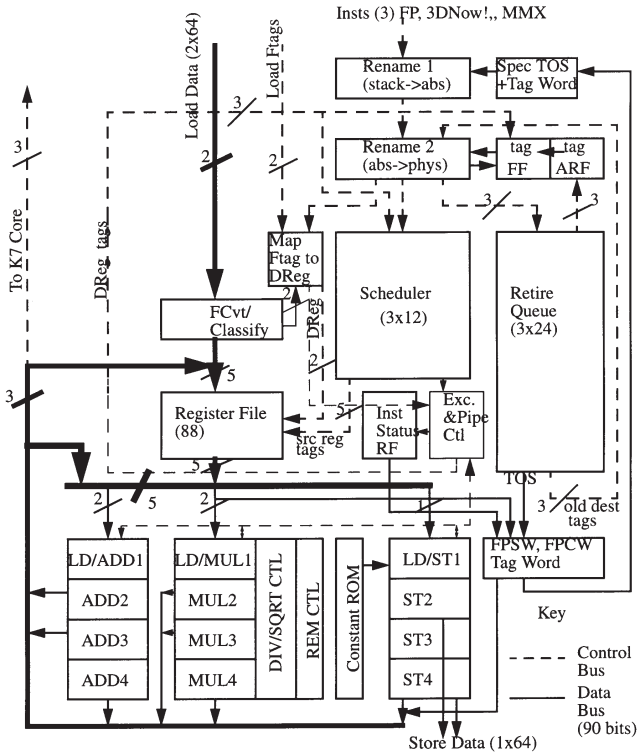

Figure 5.5.1: FPU microarchitecture.

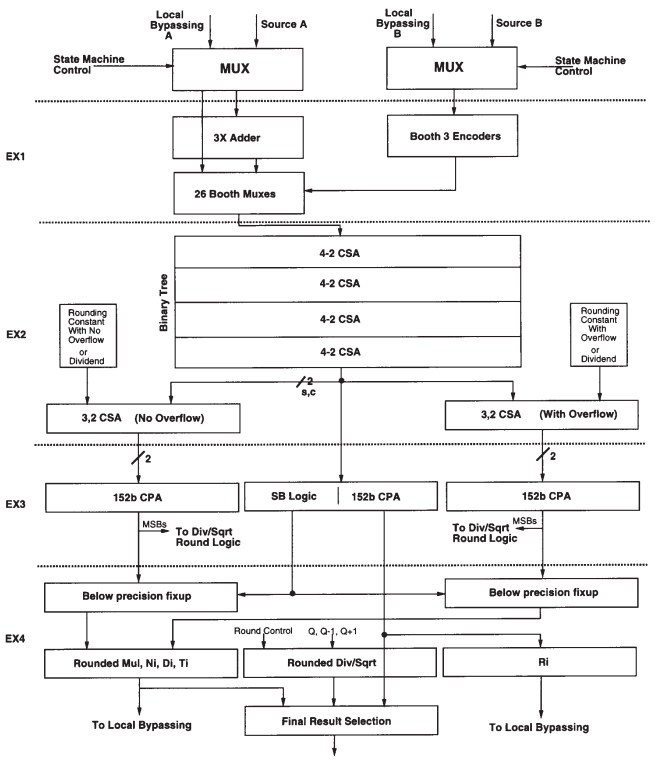

Figure 5.5.3: Multiplier pipeline.

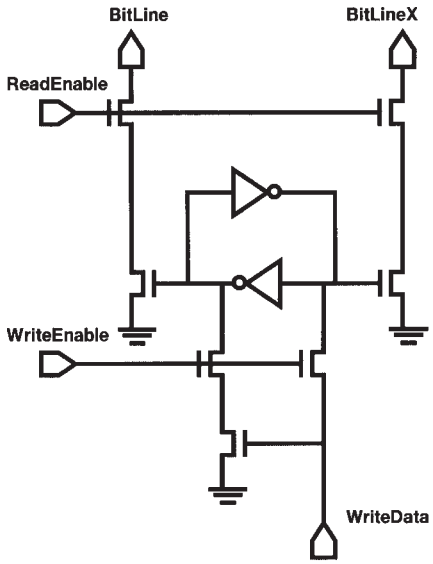

Figure 5.5.4: Register bit cell.

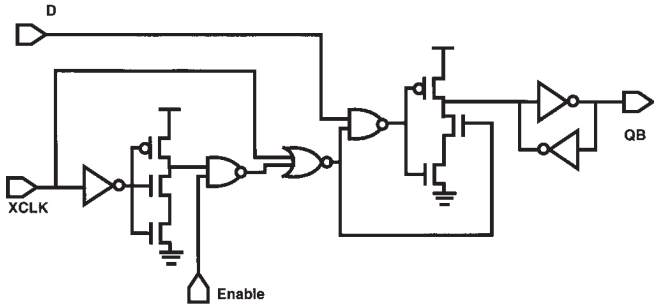

Figure 5.5.5: Basic enabled flip-flop.

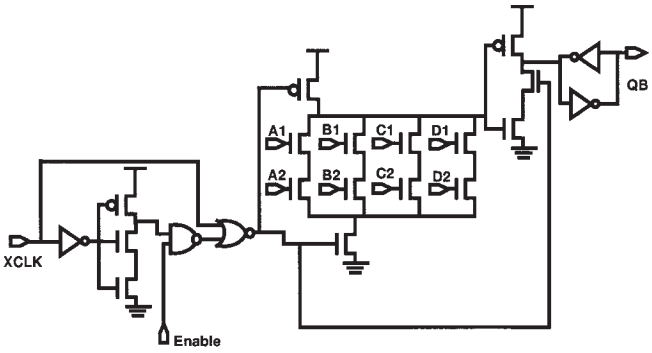

Figure 5.5.6: Four-way MUX enabled flop.

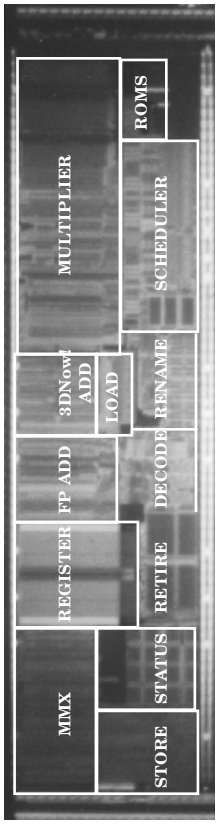

Figure 5.5.2: FPU micrograph.

Operation	Latency	Throughput
FADD	4	1
FMUL	4	1
FDIV	16/20/24	13/17/21
FSQRT	19/27/35	16/24/32
FCOM	2	1
MMX ALU	2	1
MMX Mul	3	1
3DNow! PFADD	4	1
3DNow! PFMUL	4	1
3DNow! PFRCP	3	1

Table 5.5.1: Instruction latencies.

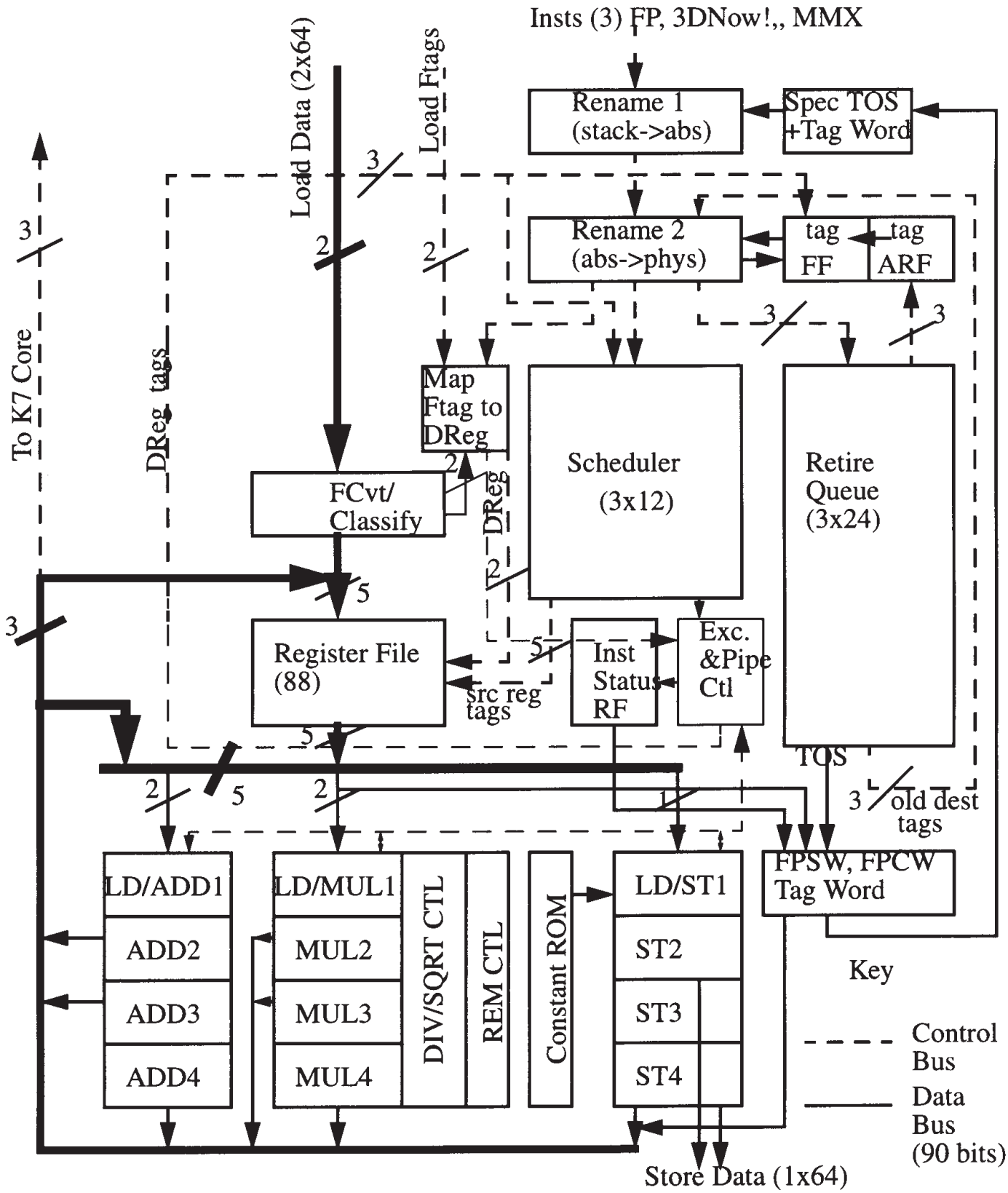

Figure 5.5.1: FPU microarchitecture.

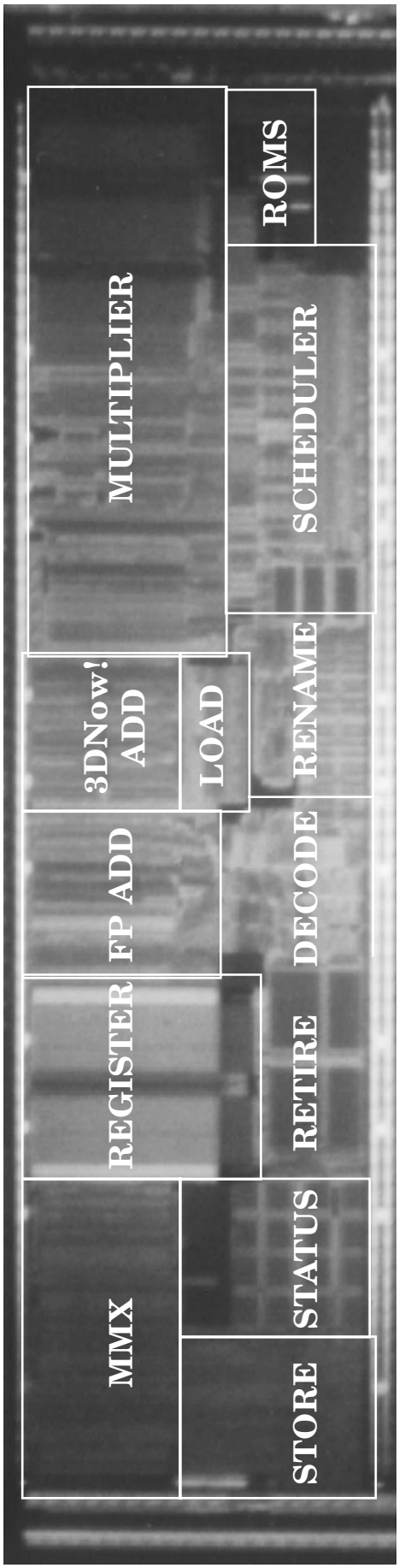

Figure 5.5.2: FPU micrograph.

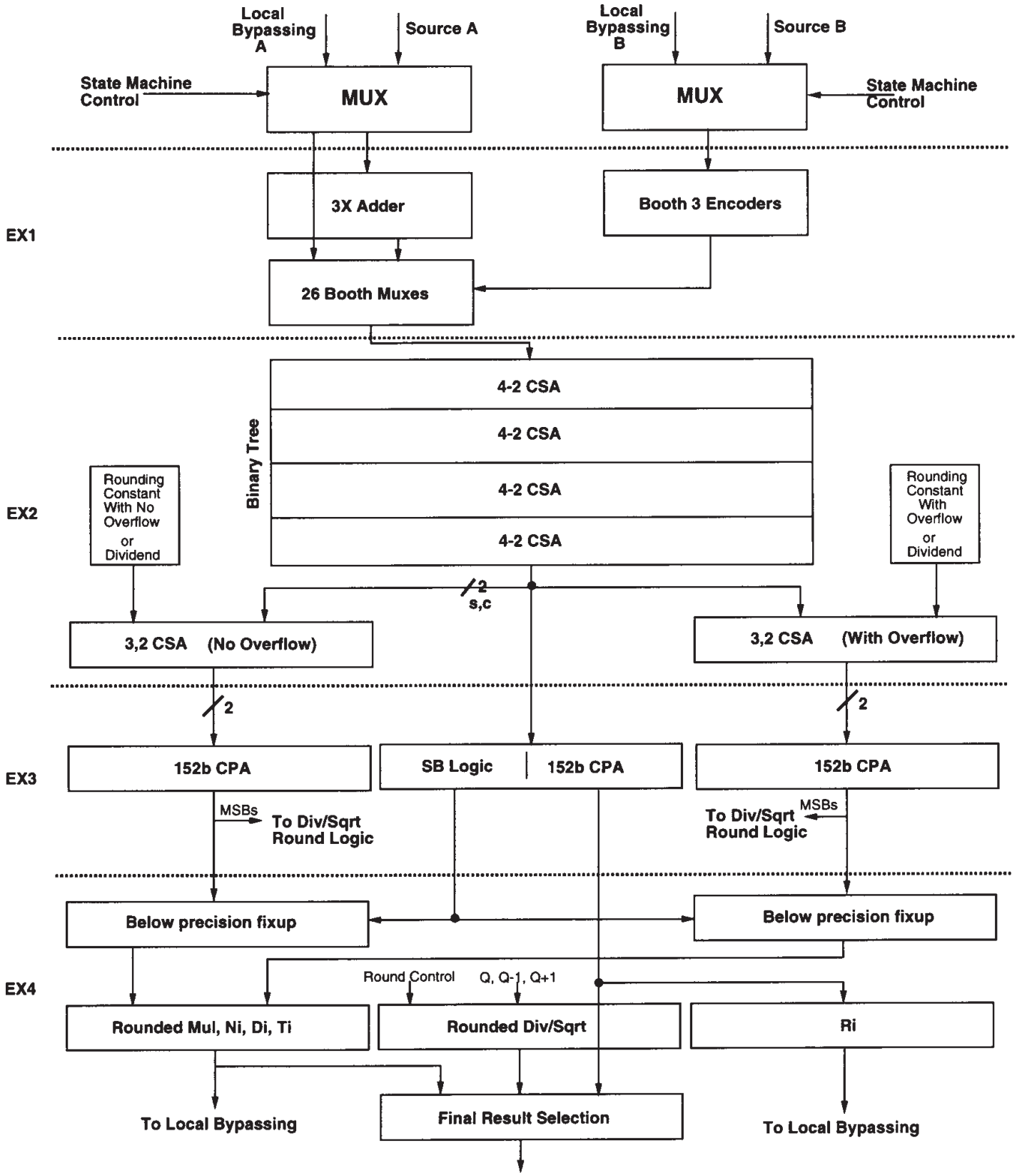

Figure 5.5.3: Multiplier pipeline.

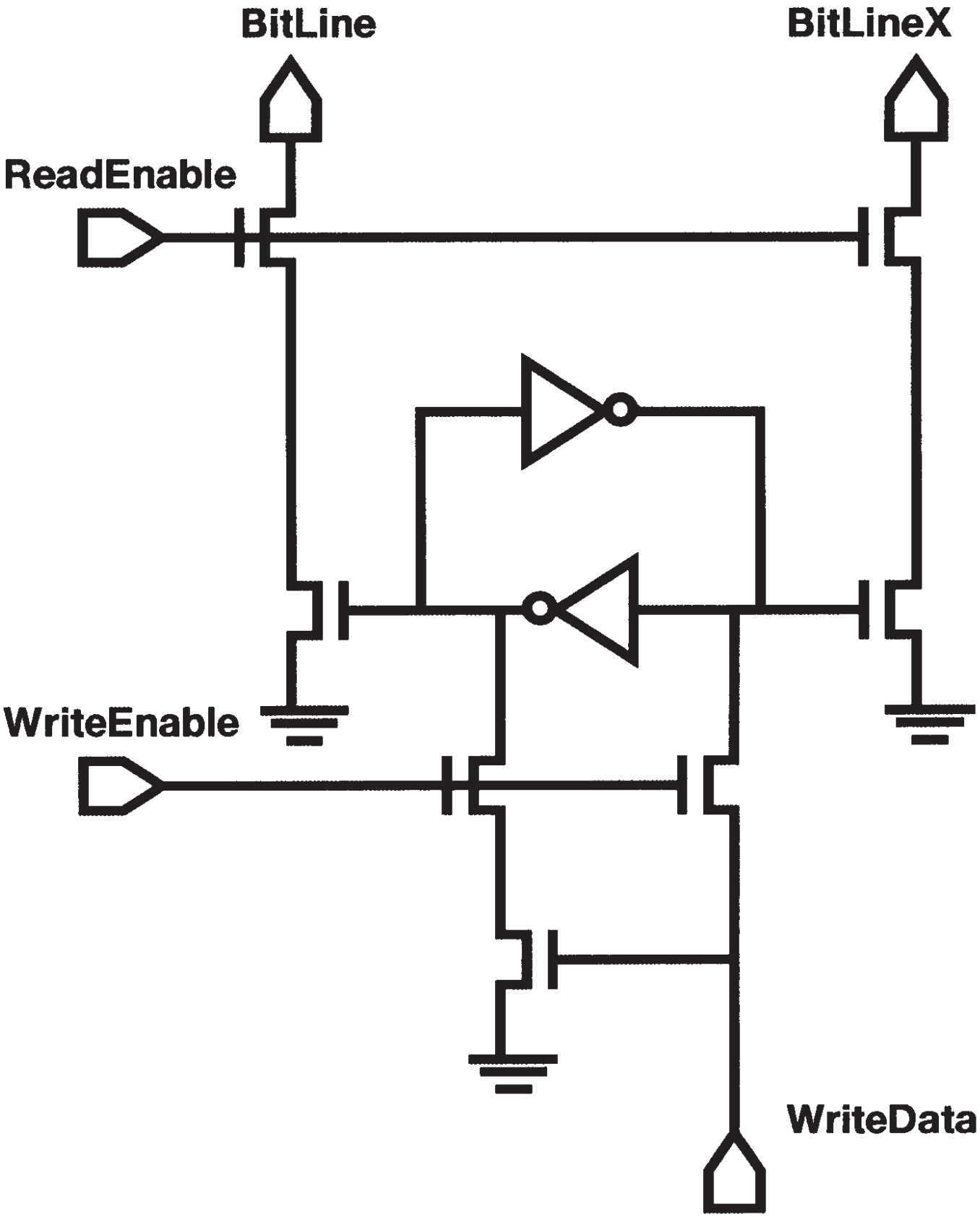

Figure 5.5.4: Register bit cell.

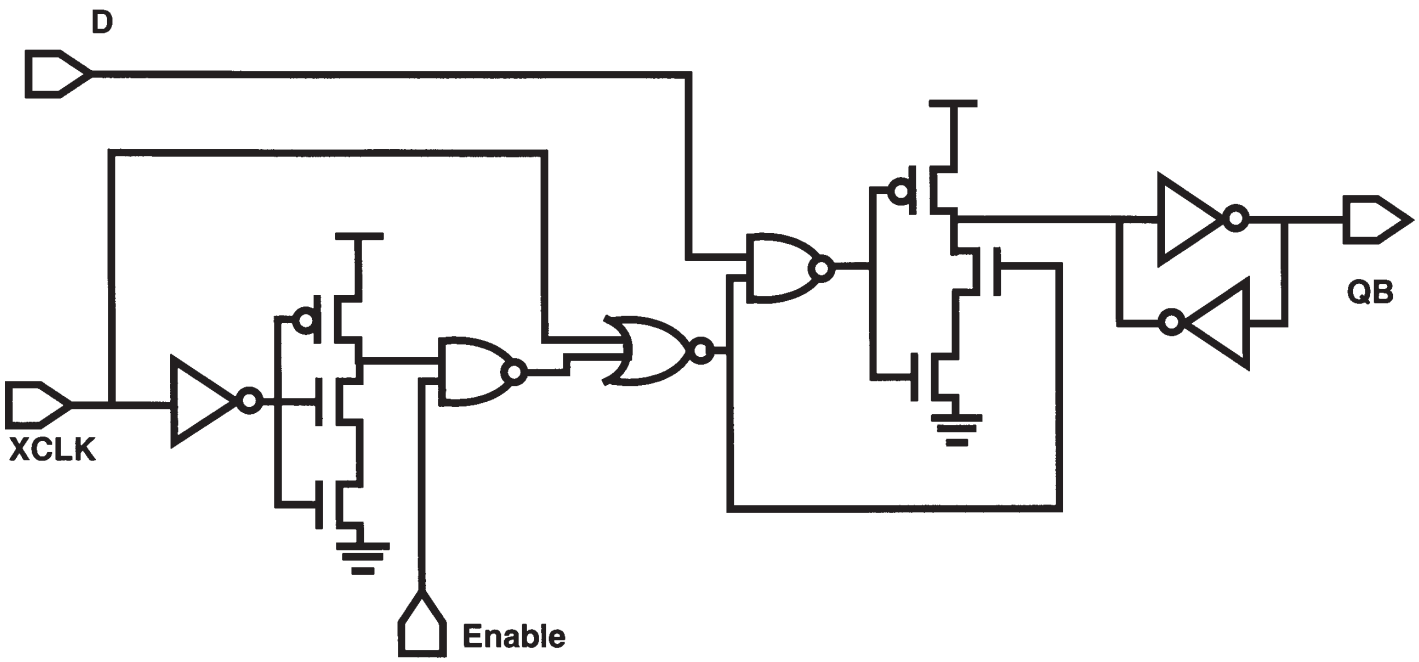

Figure 5.5.5: Basic enabled flip-flop.

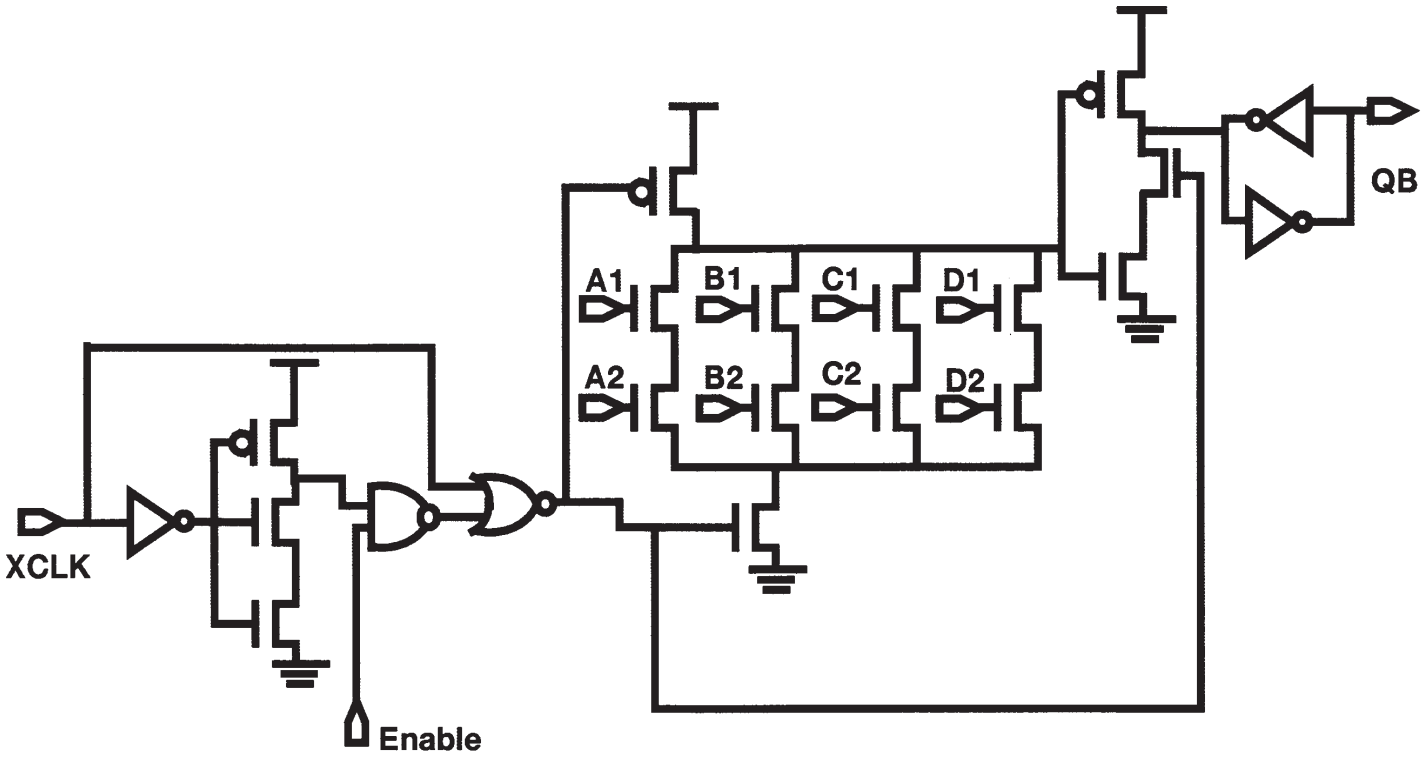

Figure 5.5.6: Four-way MUX enabled flop.

Operation	Latency	Throughput
FADD	4	1
FMUL	4	1
FDIV	16/20/24	13/17/21
FSQRT	19/27/35	16/24/32
FCOM	2	1
MMX ALU	2	1
MMX Mul	3	1
3DNow! PFADD	4	1
3DNow! PFMUL	4	1
3DNow! PFRCP	3	1

Table 5.5.1: Instruction latencies.

